Методичка 9.2 - Знакомство с фреймворком radare2. Часть 1.

Состав фреймворка radare2

rasm2 - дизассемблер
rabin2
rahash2
radiff2
rafind2
ragg2
rax2
radeco - декомпилятор

Давайте начнем с того, что такое radare2.

radare2 - это свободный кроссплатформенный фреймворк для реверс-инжиниринга, написан он на языке Си. Фреймворк включает в себя дизассемблер, шестнадцатеричный редактор, анализатор кода и еще много всего интересного.

По сути, фреймворк radare2 это набор утилит и библиотек. Наиболее интересные представлены на слайде.

Самая главная утилита фреймворка это rasm2 - это дизассемблер. Он позволяет дизассемблировать не только отдельные бинарные файлы, но и отдельные байты в виде строки.

Очень полезная утилита rabin2. Она позволяет получить различную информацию из PE- или ELF-файла. Например, импортируемые функции, библиотеки, секции, в общем, аналог Pappy, только в Linux.

Иногда в процессе анализа программы нужно посчитать хэш от какого-нибудь набора данных и алгоритм этого хэша может быть очень экзотическим. На помощь в этом нам приходит утилита rahash2.

Для того, чтобы выполнить сравнение двух файлов и увидеть разницу, мы можем воспользоваться утилитой radiff2.

С помощью утилиты rafind2 мы можем выполнить поиск строк в файле и при этом использовать регулярные выражения.

Утилита rax2 позволяет выполнять конвертацию данных в различные форматы. Например, перевести ASCII коды в буквы или перевести число из одной системы счисления в другую.

radeco - это декомпилятор.

Рекомендации по установке фреймворка radare2

Ссылка: https://github.com/radare/radare2.git

```
$ git clone https://github.com/radare/radare2.git
$ cd radare2
$ ./sys/install.sh
$ r2 -v
radare2 3.7.0-git 22267 @ linux-x86-32 git.3.6.0-99-gc1abcbe
commit: c1abcbe5e30b9050d8addbe3e9a5bb2d6fe548c3 build: 2019-07-07__05:29:01
```

Обновления появляются достаточно часто, поэтому я рекомендую ежедневно их проверять с помощью следующей команды.

```
$ ./sys/install.sh
```

Обзор основных возможностей

Исходный код программы prog-7.2

…

Компиляция программы.

```
$ gcc prog-7.2.c -o prog-7.2 -fno-stack-protector -z execstack
```

Для того, чтобы познакомиться с основными возможностями radare2 мы выполним реверс-инжиниринг программы prog-7.2.

Первое, с чего стоит начать, это собрать как можно больше различной информации о файле prog-7.2. С этим нам поможет утилита rabin2.

```
$ rabin2 -I prog-7.2

arch     x86
baddr    0x8048000
...
bintype  elf
bits     32
...
class    ELF32
...
endian   little
```

```
havecode        true
...
lang            c
...
```

Видим, что это исполняемый файл, который имеет ELF-заголовок. Файл является 32-битным и содержит в себе отладочные символы.

Используя флаг -e, мы можем получить адрес точки входа.

```
$ rabin2 -e prog-7.2
...
vaddr=0x08048370
…
```

Теперь мы знаем, что это исполняемый файл. Давайте попробуем его запустить.

```
$ prog-7.2

Usage: ./prog-7.2 <license key>

$ prog-7.2 blabla

ERROR. License key is incorrect.
```

Видим, что программа требует ввести ключ. Наша задача составить ключ, который пройдет все проверки.

Пришло время дизассемблировать программу prog-7.2.

```
$ r2 prog-7.2
```

Мы видим адрес 0x08048370, сейчас мы на нем находимся, это адрес точки входа, который мы видели ранее.

radare не анализирует исполняемый файл автоматически при запуске, как это делает дизассемблер IDA. Для проведения автоматического анализа необходимо переоткрыть файл с параметром -A или выполнить команду “aaa”.

```
[0x08048370]> aaa
```

После завершения анализа radare2 ассоциирует найденные имена со смещениями в файле. Такие имена называются флагами. Флаги сгруппированы в пространства флагов. Вывести пространства флагов можно следующей командой:

[0x08048370]> fs

…
5 * imports
…
3 * strings
...

Мы можем посмотреть содержимое каждого пространство флагов следующей командой.

[0x08048370]> fs imports; f

Мы видим перечень функций, который импортируется из libc в исполняемый файл. Давайте посмотрим на строки.

[0x08048370]> fs strings; f

Одну строку мы уже видели, она нас уведомляла о некорректном ключе, но мы видим, что в файле есть еще одна строка, которая сообщает о корректном ключе.

Как вы возможно заметили, нам вывели не совсем строки, а созданные переменные, которые ссылаются на строки - str.ERROR._License_key_is_incorrect.. Эти переменные можно будет использовать, если мы пожелаем обратиться к этой строке.

Для того, чтобы увидеть строки в явном виде, есть специальные команды iz и izz. Первая выводит строки только из секции данных, вторая из всех секций в файле.

[0x08048370]> iz

[0x08048370]> izz

На данном этапе нам будут интересны строки, которые расположены в секции данных.

В radare есть замечательная команда, которая может вывести функцию, в которая используется нужная нам строка. Для того, чтобы указать, какая строка нас интересует, мы должны указать переменную, которая была создана для этой строки.

[0x08048370]> axt str.ERROR._License_key_is_incorrect.

Мы видим, что эта строка используется в функции main. Также можно вывести такую информацию по всем строкам.

[0x08048370]> axt @@ str.*

В данном примере выражение @@ играет роль итератора, а переменные мы указывает как str.* , т.е. все множество переменных str. Видим, что другие строки тоже находятся в функции main.

Все это время мы находились на адресе точки входа программы. Здесь важно понимать, что мы сейчас выполняем статический анализ, а не динамический. Программа сейчас не запущена. Адрес, который мы видим слева, он просто играет роль указателя в рамках исполняемого файла.

Для того, чтобы перейти по смещению, мы должны воспользоваться командой “s”, которая означает - seek.

Давайте попробуем перейти на функцию main:

[0x08048370]> s main
[0x080485d3]>

Видим, что адрес изменился. Кстати, давайте узнаем, какие еще функции есть в программе prog-7.2. Для этого нам понадобится команда afl (Analyze Functions List).

[0x080485d3]> afl

Здесь мы видим функции, которые были импортированы из динамических библиотек, функцию main и некоторые другие функции. Для того, чтобы посмотреть содержимое этих функций, нам нужно выполнить дизассемблирование, но мы это сделаем в следующем видео.